

SYNTHESIS AND EQUIVALENCE OF CONCURRENT SYSTEMS

Björn LISPER

NADA, The Royal Institute of Technology, S-100 44 Stockholm, Sweden

Abstract. A framework for synthesis of synchronous concurrent systems with local memory is developed. Given an output specification of the system a *cell action structure* can be derived. This structure can be mapped into a *communication structure*, a model of the events in the target hardware with constraints on the communication possible between events, giving a schedule for the cell actions. Communication structures are interesting in their own right, and transformations defined on such can be used for showing equivalence between different computational networks. As an example, the equivalence between two specific communication structures is proved and it is shown that an FFT algorithm can be implemented on them.

1. Introduction

The design of special-purpose hardware is a field that has gained attention for a number of years. The increased possible complexity of such systems has led to hierarchical design philosophies, where the hardware is conceptually broken down into a network of communicating subpieces, thereby reducing the complexity of designing each subpiece. Notable is the concept of *systolic arrays* [10]. On the other hand these design philosophies give rise to the issue of organizing such networks of subpieces in a way that preserves correctness of operation while maximizing the total efficiency (loosely speaking throughput/hardware size). On this level of description the emphasis is on topology and timing rather than geometry.

Several attempts have been made to describe the parallel scheduling of algorithms as a mapping of cell actions (or corresponding objects) into a *space-time*. The earliest work in the area known to the author is the work concerned with parallelizing FORTRAN DO-loops for execution on multiprocessor systems [9, 12, 22]. Later it was discovered that the methods developed there could be applied to synthesis of regular hardware structures [20, 21]. In these works the event structure is somewhat obscured by the fact that the semantics are given by sequential constructs in an imperative language. Other papers give a more event-oriented description of the task to be performed, in the form of sets of recurrence relations or multivalued assignments [2, 3, 4, 15], or using a more graph-theoretic approach [13, 14, 24]. All these papers have in common that they consider fixed sets of cell actions. Designs are altered by changing the mapping from cell actions to space-time. This is the only means of improving system performance.

In this paper we will proceed a step further and derive the cell actions, given a specification of the desired outputs of the system as *polynomials* over some algebra in the inputs. This gives precise semantics to the cell actions, and it also suggests how different sets of cell actions can be derived while leaving the outputs invariant, giving an additional tool of improving the performance of the resulting system. We also augment the concept of space-time by introducing *communication orderings* on such, giving *communication structures* describing the communication pattern of distributed architectures. An equivalence relation between communication structures is defined, which can be used to show equivalence between different multiprocessor organizations.

Some proofs are just outlined or, if they are lengthy and not very interesting, even omitted. They are all given in full detail in [17].

2. Preliminaries

In this paper we will consider *heterogenous algebras* $\langle \mathcal{S}; F \rangle$ [1], where $\mathcal{S} = (S_i | i \in I')$ is a family of sets and F is a family of finitary operators.

Polynomials are functions built up recursively using operators in F [6]. If $\mathcal{A} = \langle \mathcal{S}; F \rangle$ is an algebra and $(A_x | x \in X)$ is a family of sets such that $\{A_x | x \in X\} \subseteq \{S_i | i \in I'\}$, then $\mathcal{P}^{(X)}(\mathcal{A})$, the *polynomials in X over \mathcal{A}* , is the set of functions from $\prod(A_x | x \in X)$ to some $A \in \{S_i | i \in I'\}$ given by (1), (2) below:

- (1) the *projections* e_x^X , all $x \in X$ (see [6]), all belong to $\mathcal{P}^{(X)}(\mathcal{A})$;
- (2) for all $f \in F$ and for all $p_1, \dots, p_{n(f)}$ in $\mathcal{P}^{(X)}(\mathcal{A})$ and of “right sort” (see [16]), $f \circ \langle p_1, \dots, p_{n(f)} \rangle \in \mathcal{P}^{(X)}(\mathcal{A})$ (“ \circ ” denotes composition of a function with a tuple of functions).

$n(f)$ is the arity of f . Constants are expressed as nullary operators.

$\mathcal{E}^{(X)}(\mathcal{A})$, the set of *formal expressions in X over an algebra \mathcal{A}* , is defined similarly to polynomials (cf. [6]): every $x \in X$ is a formal expression. If $f \in F$ and if $p_1, \dots, p_{n(f)} \in \mathcal{E}^{(X)}(\mathcal{A})$ are of “right sort”, then $f \bullet \langle p_1, \dots, p_{n(f)} \rangle \in \mathcal{E}^{(X)}(\mathcal{A})$. The expressions of form $f \bullet \langle p_1, \dots, p_{n(f)} \rangle$ are called *compound expressions* when $n(f) > 0$. The other expressions are called variables or constants as the corresponding polynomials.

The basic difference between formal expressions and polynomials is the notion of equality. Since polynomials are functions, equality between polynomials is the same as equality between functions. Equality between expressions, on the other hand, is defined by

- (E1) for all $x, y \in X$, $x = y$ regarded as expressions if and only if they are equal in X ;
- (E2) for all variables x and compound expressions \mathbf{p} , $x \neq \mathbf{p}$;
- (E3) if two expressions $f \bullet \langle \mathbf{p}_1, \dots, \mathbf{p}_{n(f)} \rangle, f' \bullet \langle \mathbf{p}'_1, \dots, \mathbf{p}'_{n(f')} \rangle$ are equal, then $f = f'$ and, for $i = 1, \dots, n(f)$, $\mathbf{p}_i = \mathbf{p}'_i$.

Thus two expressions are equal if and only if they are constructed in the same way. Since $\mathcal{E}^{(X)}(\mathcal{A})$ is defined recursively, we can prove properties of its elements

by induction on its structure. To do this we use the following *induction principle* for expressions (cf. [6, 18]): for every predicate q on $\mathcal{E}^{(X)}(\mathcal{A})$ it holds that IF

- (1a) for all $x \in X$, $q(x)$, and
- (1b) for all constants $f \bullet \langle \rangle$, $q(f \bullet \langle \rangle)$, and
- (2) for all compound expressions $f \bullet \langle \mathbf{p}_1, \dots, \mathbf{p}_{n(f)} \rangle$,

$$\forall i \in \{1, \dots, n(f)\} \quad [q(\mathbf{p}_i)] \Rightarrow q(f \bullet \langle \mathbf{p}_1, \dots, \mathbf{p}_{n(f)} \rangle),$$

THEN, for all $\mathbf{p} \in \mathcal{E}^{(X)}(\mathcal{A})$, $q(\mathbf{p})$.

φ , the *natural homomorphism* $\mathcal{E}^{(X)}(\mathcal{A}) \rightarrow \mathcal{P}^{(X)}(\mathcal{A})$, is defined by

- (1) $\varphi(x) = e_x^X$, all $x \in X$;
- (2) for all compound expressions $f \bullet \langle \mathbf{p}_1, \dots, \mathbf{p}_{n(f)} \rangle$ in $\mathcal{E}^{(X)}(\mathcal{A})$,

$$\varphi(f \bullet \langle \mathbf{p}_1, \dots, \mathbf{p}_{n(f)} \rangle) = f \circ \langle \varphi(\mathbf{p}_1), \dots, \varphi(\mathbf{p}_{n(f)}) \rangle.$$

The following notation will be used for equivalence relations and partitions: ε_f denotes the equivalence relation induced by f . If Π is a partition of the set A , then φ_Π denotes the natural mapping $A \rightarrow \Pi$ that maps every element of A into the block of Π where it is contained.

3. Output specification

We are interested in implementing systems that evaluate functions from inputs to outputs. Therefore, the following definition.

Definition 3.1. Let X be a countable index set. An *output specification* in X is a nonempty, countable set of functions O , where each $f \in O$ is a function $f: \prod (A_x | x \in X) \rightarrow A_f$. Every $f \in O$ is dependent on only a finite number of inputs. $\prod (A_x | x \in X)$ is the *domain* of O .

A system given by an output specification will usually be implemented using more primitive functions than the ones in the specification. This naturally leads to the study of algebras where these primitive functions are operators. For an algebra \mathcal{A} , the polynomials in $\mathcal{P}^{(X)}(\mathcal{A})$ are exactly the functions with inputs X we can realize using the operators of \mathcal{A} . Therefore it is interesting to know if we can use polynomials in $\mathcal{P}^{(X)}(\mathcal{A})$ to somehow meet a given output specification.

Definition 3.2. Let \mathcal{A} be an algebra and let O be an output specification with domain $A = \prod (A_x | x \in X)$. O is *satisfiable* in \mathcal{A} iff there exists $\eta: O \times A \rightarrow \mathcal{P}^{(X)}(\mathcal{A})$ such that, for all $f \in O$ and for all $a \in A$, $f(a) = \eta(f, a)(a)$. η is a *representation function* of O with respect to \mathcal{A} and $\eta(O, A)$ is a *polynomial representation* of O in \mathcal{A} .

Definition 3.2 gives the condition for the functions of an output specification O to be realizable by a set of polynomials. However, which polynomial corresponds to a given function f in O might vary with the inputs; that is, the structure of the way of computing f using the operators in \mathcal{A} may be *data-dependent*. An interesting case is when f always corresponds to the same polynomial.

Definition 3.3. Let \mathcal{A} be an algebra and let O be an output specification satisfiable in \mathcal{A} . $f \in O$ is *structurally data-independent* in \mathcal{A} iff there exists a representation function η of O with respect to \mathcal{A} such that, for all $a, a' \in A$, it holds that $\eta(f, a) = \eta(f, a')$.

Proposition 3.4. f structurally data-independent in $\mathcal{A} \Leftrightarrow f \in \mathcal{P}^{(X)}(\mathcal{A})$.

Proof. A structurally data-independent function f is equal to the function $\eta(f, a) \in \mathcal{P}^{(X)}(\mathcal{A})$ (which is the same polynomial regardless of a). \square

Thus, structural data-independence for a function in an algebra \mathcal{A} is the same as the function being a polynomial over \mathcal{A} . We will later see that structural data-independence of a function f makes an a priori scheduling of the evaluation of f possible.

4. Formal expressions as schemes for evaluation

Why do we consider formal expressions? The reason is that they can be given a particular interpretation; *compound expressions can be seen as representing the application of the outermost operator to the intermediate data produced by the immediate subexpressions, variables as making an input value available and constants as making a constant value available*. Thus an expression \mathbf{p} together with its subexpressions constitutes a scheme for computing $\varphi(\mathbf{p})$.

Definition 4.1. Let O be an output specification in X with domain A , let $E \subseteq \mathcal{E}^{(X)}(\mathcal{A})$ and let η be a representation function of O with respect to \mathcal{A} .

- (1) E is an *evaluable set of expressions* if and only if for all $\mathbf{p} \in E$

$$\mathbf{p} = f \bullet \langle \mathbf{p}_1, \dots, \mathbf{p}_{n(f)} \rangle \Rightarrow \mathbf{p}_1, \dots, \mathbf{p}_{n(f)} \in E.$$

- (2) The *evaluable closure* $\text{ec}(E)$ of E is the smallest evaluable set of expressions such that $E \subseteq \text{ec}(E)$.
- (3) If $\eta(O, A) \subseteq \varphi(E)$, then E is an *expression representation* of O and $\text{ec}(E)$ is a *computational scheme* for O .

For all expressions in an evaluable set of expressions E , all their subexpressions are also in E . In the following we will be interested in computational schemes for

output specifications. Informally speaking, a computational scheme contains every step, expressed as operators in the algebra \mathcal{A} applied to subexpressions, necessary to compute each polynomial in the polynomial representation of the output specification. Note that there in general may be many computational schemes for a given output specification O .

Definition 4.2. The binary relation $<_e$ on $\mathcal{E}^{(X)}(\mathcal{A})$ is given by the following: Let $\mathbf{p}', \mathbf{p} \in \mathcal{E}^{(X)}(\mathcal{A})$. $\mathbf{p}' <_e \mathbf{p}$ iff $\mathbf{p} = f \bullet \langle \mathbf{p}_1, \dots, \mathbf{p}_{n(f)} \rangle$, and, for some $k \in \{1, \dots, n(f)\}$, $\mathbf{p}' = \mathbf{p}_k$.

Under our interpretation of expressions as representing events where operators are applied to intermediate data, the relation $<_e$ defined above obtains a special meaning: $\mathbf{p}' <_e \mathbf{p}$ means that *the result of evaluating \mathbf{p}' is used as input when evaluating \mathbf{p}* . Thus $<_e$ can be considered a *data dependence relation* on $\mathcal{E}^{(X)}(\mathcal{A})$.

Let us call the binary relation $<$ on the set A *finite-downward* iff $<$ is a strict partial order on A and for all $a \in A$ it holds that $\{a' \mid a' < a\}$ is finite. Then the following can be stated.

Theorem 4.3. $<_e^+$ is finite-downward.

Proof (sketch). It is easily seen that the induction principle for expressions implies that *the well-founded induction principle* (cf. [19]) holds for $<_e$. Thus $<_e$ is well-founded. It then suffices to show that for all $\mathbf{p} \in \mathcal{E}^{(X)}(\mathcal{A})$ there are only finitely many \mathbf{p}' such that $\mathbf{p}' <_e \mathbf{p}$. This follows since every operator has finite arity. \square

$<_e^+$ determines the amount of parallelity that is possible when evaluating a set of expressions. If $\mathbf{p}' <_e^+ \mathbf{p}$, the evaluation of \mathbf{p}' has to take place before the one of \mathbf{p} . If $\mathbf{p}' \not<_e^+ \mathbf{p}$ and $\mathbf{p} \not<_e^+ \mathbf{p}'$, they may take place concurrently. Theorem 4.3 assures that any evaluation of an expression has to be preceded by only a finite number of other evaluations of expressions. Thus the evaluation of any expression can be scheduled in finite time. We can also note that $\langle E, <_e^* \rangle$ is an *elementary event structure* (cf. [23]).

5. Evaluating expressions several times

In the previous section we considered expressions as representing events of computation, where the outermost operator of the expression was applied to the intermediate data produced by the immediate subexpressions. However, given a set of expressions there is a one-to-one correspondence between the expressions in it and the events of their evaluation. We want to add the possibility of evaluating the same expression several times. Therefore the concept of *computational events* is introduced.

Definition 5.1. Let D be a countable set and E a computational scheme for an output specification O . If there exists a function $\psi: D \rightarrow E$ that is onto, we say that D is a *set of computational events of E* .

Thus every computational event p is labelled with an expression $\psi(p)$ and p represents an evaluation of $\psi(p)$ in the computational scheme of O .

Given a set of computational events D of E we are interested in finding data dependence relations on D that are consistent with $<_e$ on E . That is, since $<_e$ describes the transmission of data necessary to evaluate all expressions in E , the relation on D should describe the transmission of data necessary to evaluate all computational events of expressions in D . There are in general several possibilities to accomplish that. Therefore we define not only one relation, but a whole class of relations on D satisfying the above. The definition is based on the following observation: Given a computational event p of a compound expression $\psi(p)$, for each p' such that $p' <_e \psi(p)$ there may be several computational events p' such that $\psi(p') = p'$. Therefore, for each argument of $\psi(p)$, exactly one of the computational events of the corresponding subexpression must be selected to provide the input to p .

Definition 5.2. Let $\psi: D \rightarrow E$ be onto. $\text{comp}(D, \psi)$ is the set of computational events p in D such that $\psi(p)$ is a compound expression $f \bullet \langle p_1, \dots, p_{n(f)} \rangle$ and $n(f) > 0$.

$\text{SF} = (s_p \mid p \in \text{comp}(D, \psi))$ is a *family of selector functions with respect to D and ψ* iff, for every p in $\text{comp}(D, \psi)$ where $\psi(p) = f \bullet \langle p_1, \dots, p_{n(f)} \rangle$, s_p is a function $\{1, \dots, n(f)\} \rightarrow D$ such that $\psi(s_p(i)) = p_i$ for all $i \in \{1, \dots, n(f)\}$.

Definition 5.3. Let $\psi: D \rightarrow E$ be onto. For every family of selector functions $\text{SF} = (s_p \mid p \in \text{comp}(D, \psi))$, $<_{\text{SF}}$ is defined by

- (1) for all p in $\text{comp}(D, \psi)$ and for all $i \in \{1, \dots, n(f)\}$, $s_p(i) <_{\text{SF}} p$.
- (2) for no other p', p in D , $p' <_{\text{SF}} p$.

The *class of data dependence relations on D with respect to ψ and E* , $\mathcal{R}(D, \psi, E)$, is the set $\{<_{\text{SF}} \mid \text{SF} \text{ is a family of selector functions with respect to } D \text{ and } \psi\}$.

For any $< \in \mathcal{R}(D, \psi, E)$, $\langle D, < \rangle$ is a *computational event structure of E under ψ* .

The selector function s_p tells us for every argument of $\psi(p)$ which computational event that is going to provide it, and there is a data dependency between exactly those computational events and p . The following theorem shows that the property proved in Theorem 4.3 for data dependence relations on sets of expressions also hold for data dependence relations on sets of computational events:

Theorem 5.4. For every $< \in \mathcal{R}(D, \psi, E)$, $<^+$ is finite-downward.

Proof (sketch). For all $p, p' \in D$ it holds that $p < p' \Rightarrow \psi(p) <_e \psi(p')$. Since $<_e$ is well-founded, it follows (see, for instance, [19]) that $<$ is well-founded. It then

suffices to show that for all $p' \in D$ there are only finitely many p such that $p < p'$. This follows since the top operator of p' has finite arity and exactly one computational event is selected for each argument by the selector function of p' . \square

We can also note that $\langle D, <^* \rangle$ is an elementary event structure for every $< \in \mathcal{R}(D, \psi, E)$.

6. Cell actions

So far, we have associated exactly one expression with one event, indicating the application of the outermost operator of that expression at that event. However, it is in many cases practical to consider events that correspond to the evaluation of more than one expression. This leads to the concept of *cell actions*, which essentially are blocks in a partition fulfilling certain restrictions of a set of computational events. Before giving a strict definition of cell actions, we give the following technical definition.

Definition 6.1. Let A, B be sets, let $f: A \rightarrow B$ and let ε_f be the equivalence relation on A induced by f . Let R_A be a binary relation on A and R_B a binary relation on B . $\langle B, R_B \rangle$ is *finitely inherited from* $\langle A, R_A \rangle$ under f , or $\text{fi}(\langle A, R_A \rangle, \langle B, R_B \rangle, f)$, iff

- (fi1) $\forall a \in A [a]_{\varepsilon_f}$ is finite;
- (fi2) $\forall b, b' \in B [b R_B b' \Leftrightarrow (b \neq b' \wedge \exists a, a' \in A [a R_A a' \wedge f(a) = b \wedge f(a') = b'])]$;
- (fi3) $\langle A, R_A \rangle$ finite-downward $\Rightarrow \langle B, R_B \rangle$ finite-downward.

The meaning of fi becomes more evident if the partition induced by f is considered, as shown in Fig. 1. If we consider the structures as directed graphs, the effect of (fi2) is to extract the “external” edges of R_A into R_B . Edges internal to a block are hidden since $b R_B b' \Rightarrow b \neq b'$.

Lemma 6.2 (fi-transitivity). *fi is transitive with respect to its first two arguments; that is,*

$$\text{fi}(\langle A, R_A \rangle, \langle B, R_B \rangle, f) \wedge \text{fi}(\langle B, R_B \rangle, \langle C, R_C \rangle, g) \Rightarrow \text{fi}(\langle A, R_A \rangle, \langle C, R_C \rangle, g \circ f).$$

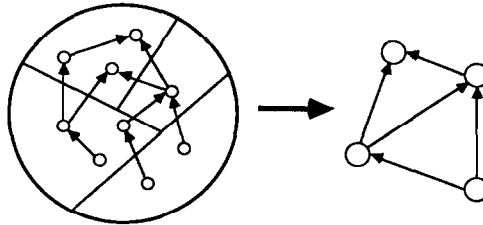


Fig. 1. $\text{fi}(\langle A, R_A \rangle, \langle B, R_B \rangle, f)$.

Proof. See [17]. \square

Lemma 6.3 (fi-uniqueness)

$$\text{fi}(\langle A, R_A \rangle, \langle R, R_B \rangle, f) \wedge \text{fi}(\langle A, R_A \rangle, \langle B, R'_B \rangle, f) \Rightarrow R_B = R'_B.$$

Proof. The lemma follows immediately from (fi2) for $\langle A, R_A \rangle, \langle B, R_B \rangle$ and $\langle A, R_A \rangle, \langle B, R'_B \rangle$ in Definition 6.1. \square

Lemma 6.4. *If $\text{fi}(\langle A, R_A \rangle, \langle B, R_B \rangle, f), R'_A \subseteq R_A$ and R_B^+ is finite-downward, then there exists an R'_B such that $\text{fi}(\langle A, R'_A \rangle, \langle B, R'_B \rangle, f)$ and $R'_B \subseteq R_B$.*

Proof. See [17]. \square

Lemma 6.5. *If R_A is irreflexive and if f is a bijection $A \rightarrow B$, then there exists a relation R_B on B such that $\text{fi}(\langle A, R_A \rangle, \langle B, R_B \rangle, f)$ and f is a graph isomorphism $\langle A, R_A \rangle \rightarrow \langle B, R_B \rangle$.*

Proof. See [17]. \square

We are now ready to define what we mean by sets of cell actions.

Definition 6.6. Let $\langle D, < \rangle$ be a computational event structure. Let C be a partition of D . $\langle C, <_c \rangle$ is a *cell action structure derived from* $\langle D, < \rangle$, and $<_c$ is the *immediate precedence relation on* C iff $\text{fi}(\langle D, < \rangle, \langle C, <_c \rangle, \varphi_C)$ (φ_C is the natural mapping $D \rightarrow C$). The blocks in C are called *cell actions*. C is called a *set of cell actions*.

Cell actions are aimed to model events of computation in a network of processing elements, or cells. From (fi2) we can conclude that $c <_c c'$ iff $c \neq c'$ and there exist computational events $p \in c, p' \in c'$ such that $p < p'$. Since $<$ describes the data transmission between computational events, $<_c$ describes the data transmission between cell actions. Data transfers internal to a cell action are hidden. We can also note that $<_c^+$ is finite-downward since $<$ is finite-downward and that $\langle C, <_c^* \rangle$ is an elementary event structure.

Let us finally mention that it is possible, given a set of cell actions C , to formally derive a *set of multiple-assignments* representing C , where each assignment corresponds to a cell action in C . The details are worked out in [17]. This relates the approach taken here to the ones in [2, 3, 4, 15].

7. Space-time

Space-time is a model of the events in a network of cells; every space point is a possible location for a cell and at every time something (for instance, a cell action)

may take place at a cell. The basic idea in this paper is to consider *mappings* from sets of cell actions into space-time, assigning one space-time coordinate to every cell action telling where and when it is to be performed. From this schedule the minimal hardware capable of supporting the execution of the set of cell actions can be derived. Let us first define what we mean by a space-time.

Definition 7.1 (*Space-time*)

- (st1) Any countable nonempty set R is a *space*.
- (st2) For any *least time* $t_0 \in \mathbb{Z}$, $T = \{n + t_0 \mid n \in \mathbb{N}\}$ is a *set of times*.
- (st3) If T is a set of times and R is a space, then $T \times R$ is a *space-time*.

This simple construction with a global, discrete time is a model of synchronous, clocked hardware. We are thus restricting us to the synthesis of such.

Definition 7.2. The binary relation $<_s$ on the space-time S is a *communication ordering* on S if and only if

- (co1) for all $\langle t, r \rangle, \langle t', r' \rangle$ in S , $\langle t, r \rangle <_s \langle t', r' \rangle \Rightarrow t \leq t'$.
- (co2) $<_s^+$ is finite-downward.
- $\langle S, <_s \rangle$ is called a *communication structure*.

Communication orderings are used to describe the communication constraints we want to pose on the resulting hardware. $s <_s s'$ means that communication may take place between events s and s' .

We now consider functions from a set of cell actions to a space-time. First we give a more general definition.

Definition 7.3. Let A be a set, $<$ a relation on A and S a space-time. For any $F: A \rightarrow S$, we define the *mapped relation* $<_F$ on S by $\text{fi}(\langle A, < \rangle, \langle S, <_F \rangle, F)$ if it exists.

By Lemma 6.3 $<_F$ must be unique if it exists. Therefore it is well-defined. For functions F from a set of cell actions to a space-time the *mapped precedence relation* $<_{cF}$ is of interest. The interpretation of F is that the cell action c is performed at the event $F(c)$. Thus $<_{cF}$ gives the following communication requirements: if $F(c) <_{cF} F(c')$, output data from c are used as input to c' and therefore these data must be sent between the events $F(c)$ and $F(c')$. Let us now define what kind of mappings from a set of cell actions to space-time we allow.

Definition 7.4. Let $\langle C, <_c \rangle$ be a cell action structure and $\langle S, <_s \rangle$ a communication structure. W is a *weakly correct mapping* $\langle C, <_c \rangle \rightarrow \langle S, <_s \rangle$ iff it is a function $C \rightarrow S$ such that

$$<_{cW} \text{ exists and } <_{cW} \subseteq <_s.$$

W is a *correct mapping* $\langle C, <_c \rangle \rightarrow \langle S, <_s \rangle$ iff it is a weakly correct mapping $\langle C, <_c \rangle \rightarrow \langle S, <_s \rangle$ that is 1-1 (see Fig. 2).

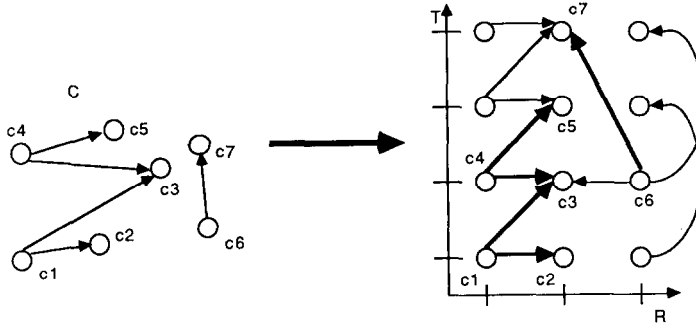


Fig. 2. A cell action structure correctly mapped into a communication structure.

A weakly correct mapping must fulfil the finiteness conditions given by Definitions 7.3 and 6.1. The monotonicity constraint $<_{cW} \subseteq <_c$ assures that the communication requirements of $\langle C, <_c \rangle$ can be met by $\langle S, <_s \rangle$ under the scheduling given by W . For a correct mapping W , there is a *unique* cell action for each space-time point in $W(C)$. This is often desirable since then every cell action c directly describes the action performed at the event $W(c)$.

Theorem 7.5. *Let $\langle C, <_c \rangle$ be a cell action structure derived from the computational event structure $\langle D, < \rangle$. Let $\langle S, <_s \rangle$ be a communication structure. For every weakly correct mapping $W: \langle C, <_c \rangle \rightarrow \langle S, <_s \rangle$, $\langle C', <'_c \rangle$, where $C' = D / \varepsilon_{W \circ \varphi_C}$, is a cell action structure derived from D , and there exists a correct mapping $M: \langle C', <'_c \rangle \rightarrow \langle S, <_s \rangle$ such that $W \circ \varphi_C = M \circ \varphi_{C'}$. The immediate precedence relation $<'_c$ on C' is as follows: for all $p, p' \in D$,*

$$\varphi_C(p) <'_c \varphi_C(p') \Leftrightarrow W \circ \varphi_C(p) <_{cW} W \circ \varphi_C(p').$$

Proof (sketch). Any function $f: A \rightarrow B$ can be decomposed as $f = f_i \circ f_o$, where f_i is the natural mapping $A \rightarrow A / \varepsilon_f$ and f_o is the function $A / \varepsilon_f \rightarrow B$ for which $f_o([a]_{\varepsilon_f}) = f(a)$ for all $a \in A$. Thus $W \circ \varphi_C$ can be decomposed into $M \circ \varphi_{C'}$ where $C' = D / \varepsilon_{W \circ \varphi_C}$ and $M: C' \rightarrow S$ is 1-1. It can be shown, using fi-transitivity (Lemma 6.2) that

$$\text{fi}(\langle D, < \rangle, \langle C', <'_c \rangle, \varphi_{C'});$$

that is, $\langle C', <'_c \rangle$ is a cell action structure derived from $\langle D, < \rangle$. Further it can be shown, using fi-transitivity and fi-uniqueness (Lemma 6.3), that $<_{cM}$ exists and $<_{cM} = <_{cW}$. Since M was found to be 1-1, this implies that M is a correct mapping $\langle C', <'_c \rangle \rightarrow \langle S, <_s \rangle$. \square

Thus it is always possible to derive a correct mapping from every weakly correct mapping, describing the same schedule. An interpretation of Theorem 7.5 is that all the old cell actions mapped to the same point in space-time are merged into a new, bigger cell action. This is illustrated in Fig. 3.

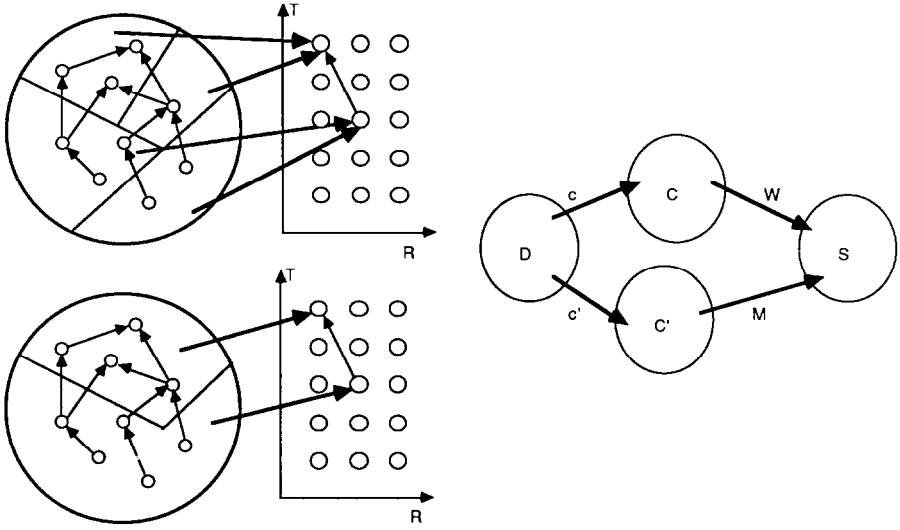


Fig. 3. Merge of cell actions according to Theorem 7.5 and a commuting diagram illustrating the theorem.

A weakly correct mapping gives a schedule for the evaluation of a set of cell actions; that is, where and when every cell action is to be performed. Now, assume that the set of cell actions is derived from a set of computational events of a computational scheme of an output specification where all functions are structurally data-independent. We will then know *in advance* which computational event(s) will yield the value of each computed function in the output specification. The times and places for these computational events are uniquely determined by the weakly correct mapping, and we will know in advance where and when to pick up each result. If, on the contrary, the output specification is structurally data-dependent, this is not possible to know before the actual inputs to the computation are available. In this case, the weakly correct mapping must schedule every possible execution leading to a result.

Some proposed design methodologies turn existing designs into new ones, see, for instance, [8, 26] or [11]. In the framework developed here the transformation of old designs to new ones is most conveniently seen as a *transformation of space-time*.

Definition 7.6. Let $\langle S, <_s \rangle$ and $\langle S', <'_s \rangle$ be communication structures. $A: S \rightarrow S'$ is a *weak space-time transformation* $\langle S, <_s \rangle \rightarrow \langle S', <'_s \rangle$ iff

$$<_{sA} \text{ exists and } <_{sA} \subseteq <'_s.$$

A weak space-time transformation $\langle S, <_s \rangle \rightarrow \langle S', <'_s \rangle$ is a *space-time transformation* $\langle S, <_s \rangle \rightarrow \langle S', <'_s \rangle$ iff it is 1-1.

Theorem 7.7. Let $\langle S, <_s \rangle$, $\langle S', <'_s \rangle$ be communication structures and let A be a weak space-time transformation $\langle S, <_s \rangle \rightarrow \langle S', <'_s \rangle$. Let $\langle C, <_c \rangle$ be a cell action structure.

Then, for every weakly correct mapping $W: \langle C, <_c \rangle \rightarrow \langle S, <_s \rangle$, $A \circ W$ is a weakly correct mapping $\langle C, <_c \rangle \rightarrow \langle S', <'_s \rangle$.

Proof (sketch). Since W is a weakly correct mapping $\langle C, <_c \rangle \rightarrow \langle S, <_s \rangle$, there is a $<_{cW} \subseteq <_c$ such that $\text{fi}(\langle C, <_c \rangle, \langle S, <_{cW} \rangle, W)$ and $<_{cW} \subseteq <_s$. Since A is a weak space-time transformation $\langle S, <_s \rangle \rightarrow \langle S', <'_s \rangle$, there is a $<_{sA} \subseteq <_s$ such that $\text{fi}(\langle S, <_s \rangle, \langle S', <_{sA} \rangle, A)$ and $<_{sA} \subseteq <'_s$. $<_{sA}^{++}$ is finite-downward, which then implies that $<_{sA}^+$ is finite-downward. By Lemma 6.4 there is a $<_{cWA} \subseteq <_{sA}$ such that $\text{fi}(\langle S, <_{cW} \rangle, \langle S', <_{cWA} \rangle, A)$, and it follows that $<_{cWA} \subseteq <'_s$. fi-transitivity now yields $\text{fi}(\langle C, <_c \rangle, \langle S', <_{cWA} \rangle, A \circ W)$. \square

Corollary 7.8. If W is a correct mapping $W: \langle C, <_c \rangle \rightarrow \langle S, <_s \rangle$ and if A is a space-time transformation $\langle S, <_s \rangle \rightarrow \langle S', <'_s \rangle$, then $A \circ W$ is a correct mapping $\langle C, <_c \rangle \rightarrow \langle S', <'_s \rangle$.

Thus, weak space-time transformations give correct schedules on the new communication structure for all cell action structures correctly scheduled by a weakly correct mapping on the first communication structure. Similar results appear in [5], where a more graph-theoretic approach is taken. Space-time transformations that are not weak are of special interest since they alter neither the internal structure of the cell actions nor the dependence structure between them:

Definition 7.9. The binary relation \rightarrow^{stt} on communication structures is defined as follows: $\langle S, <_s \rangle \rightarrow^{\text{stt}} \langle S', <'_s \rangle$ iff there exists a space-time transformation $\langle S, <_s \rangle \rightarrow \langle S', <'_s \rangle$.

$\leftrightarrow^{\text{stt}}$ is defined as follows: $\langle S, <_s \rangle \leftrightarrow^{\text{stt}} \langle S', <'_s \rangle$ iff $\langle S, <_s \rangle \rightarrow^{\text{stt}} \langle S', <'_s \rangle$ and $\langle S', <'_s \rangle \rightarrow^{\text{stt}} \langle S, <_s \rangle$.

Proposition 7.10. \rightarrow^{stt} is transitive and reflexive.

Proof (sketch). *Transitivity:* assume that $\langle S, <_s \rangle, \langle S', <'_s \rangle, \langle S'', <''_s \rangle$ are communication structures with space-time transformations $A: \langle S, <_s \rangle \rightarrow \langle S', <'_s \rangle$ and $B: \langle S', <'_s \rangle \rightarrow \langle S'', <''_s \rangle$. $B \circ A$ is then 1-1. The proof that $B \circ A$ is a space-time transformation $\langle S, <_s \rangle \rightarrow \langle S'', <''_s \rangle$ is exactly analogous to the proof of Theorem 7.7.

Reflexivity: The identity function from a space-time S to itself is always a space-time transformation $\langle S, <_s \rangle \rightarrow \langle S, <_s \rangle$ for any communication ordering $<_s$ on S . \square

Theorem 7.11. $\leftrightarrow^{\text{stt}}$ is an equivalence relation.

Proof. Transitivity, reflexivity follows from the corresponding properties for \rightarrow^{stt} . Symmetry follows directly from the definition. \square

If $\langle S, <_s \rangle \rightarrow^{\text{stt}} \langle S', <'_s \rangle$, then all sets of cell actions executable on $\langle S, <_s \rangle$ can be executed on $\langle S', <'_s \rangle$ without altering the partial order of execution. $\langle S, <_s \rangle \rightarrow^{\text{stt}} \langle S', <'_s \rangle$ means that all sets of cell actions executable on one of the

communication structures can be executed on the other, and vice versa. Confer (m, k) -simulation in [5].

Proposition 7.12. *If there exists a space-time transformation $A: \langle S, <_s \rangle \rightarrow \langle S', <'_s \rangle$ that is a bijection $S \rightarrow S'$ and if $\text{fi}(\langle S, <_s \rangle, \langle S', <'_s \rangle, A)$, then $\langle S, <_s \rangle \leftrightarrow^{\text{stt}} \langle S', <'_s \rangle$.*

Proof. Assume that A is a bijection $S \rightarrow S'$ and that $\text{fi}(\langle S, <_s \rangle, \langle S', <'_s \rangle, A)$.

$(\rightarrow^{\text{stt}})$: Follows directly.

$(\leftarrow^{\text{stt}})$: Since $<_s$ is a communication ordering, it is irreflexive. By Lemma 6.5 it follows that A is a graph isomorphism $\langle S, <_s \rangle \rightarrow \langle S', <'_s \rangle$ which in turn implies that A^{-1} is a graph isomorphism $\langle S', <'_s \rangle \rightarrow \langle S, <_s \rangle$. $<'_s$ is also a communication ordering and thus irreflexive. Lemma 6.5 applied once more now implies $\text{fi}(\langle S', <'_s \rangle, \langle S, <_s \rangle, A^{-1})$. Since A is a bijection, A^{-1} is 1-1. $\langle S, <_s \rangle \leftarrow^{\text{stt}} \langle S', <'_s \rangle$ follows. \square

8. Summary

The commuting diagram in Fig. 4 shows the way from specification to space-time structure. All symbols have their meaning from the earlier sections. Informally speaking, going from left to right means synthesis, and going in the opposite direction means verification (to show by construction that a design meets a given specification). The synthesis process does not necessarily start in the leftmost position with the output specification. A common situation is that the output specification is given implicitly by, say, a set of cell actions or an equivalent set of assignments. In this case, the process of synthesis reduces to the task of finding a proper communication structure describing the desired target hardware and to find a weakly correct mapping that gives a good utilization of it.

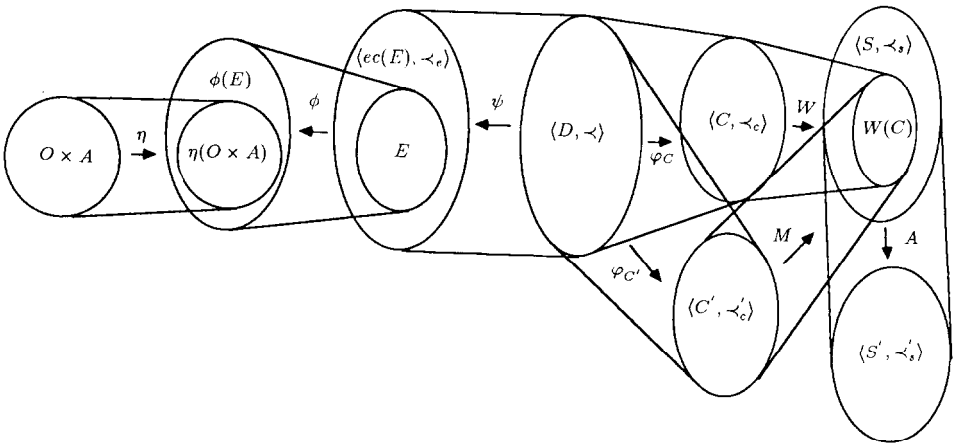


Fig. 4. Commuting diagram.

9. Example

In this example we will show $\leftrightarrow^{\text{stt}}$ -equivalence between two communication structures. The communication structures considered are the *perfect shuffle-structure* $\langle S_{\text{ps}}, <_{\text{ps}} \rangle$ describing a typical execution on the perfect shuffle network (cf. [25]) and a communication structure $\langle S_j, <_j \rangle$ describing space-time communication in a network similar to a type of Fast Fourier Transform networks (cf. [7, 8]) giving $O(\log n)$ space complexity and $O(n)$ time complexity (whereas the perfect shuffle gives the converse). We will also give a verification that a Fast Fourier Transform algorithm (FFT) can be implemented on these networks. As an intermediate vehicle we will use the *time-dependent binary cube-structure* $\langle S_{\text{tbc}}, <_{\text{tbc}} \rangle$.

The perfect shuffle-structure of order m is given by

- $S_{\text{ps}}: \mathbb{N} \times \{0, \dots, 2^m - 1\}$;
- $<_{\text{ps}}: t = 1, \dots, m+1, i = 0, \dots, 2^m - 1$:

$$\langle t-1, \lfloor \frac{1}{2}i \rfloor \rangle <_{\text{ps}} \langle t, i \rangle \quad \text{and} \quad \langle t-1, \lfloor \frac{1}{2}i \rfloor + 2^{m-1} \rangle <_{\text{ps}} \langle t, i \rangle.$$

The time-dependent binary m -cube structure is defined by

- $S_{\text{tbc}}: \mathbb{N} \times \{0, 1\}^m$;
- $<_{\text{tbc}}: t = 1, \dots, m+1, j_{m-t+1 \bmod m}, i_0, \dots, i_{m-1} \in \{0, 1\}$:

$$(t-1, i_{m-1}, \dots, j_{m-t+1 \bmod m}, \dots, i_0) <_{\text{tbc}} (t, i_{m-1}, \dots, i_{m-t+1 \bmod m}, \dots, i_0).$$

$\langle S_j, <_j \rangle$ is defined by

- $S_j: \mathbb{N} \times \mathbb{N}$
- $<_j$: let A_j be a bijection $S_{\text{tbc}} \rightarrow S_j$ such that the following holds:
 $i_0, \dots, i_{m-1} \in \{0, 1\}$:

$$A_j(0, i_{m-1}, \dots, i_0) = \left\langle \sum_{n=1}^{m-1} i_n 2^n + i_0 2^m, 0 \right\rangle;$$

$t = 1, \dots, m+1$:

$$A_j(t, i_{m-1}, \dots, i_0) = \left\langle \sum_{n=0}^{m-1} i_n 2^n + 2^{m+1} - 2^{m+1-t}, t \right\rangle.$$

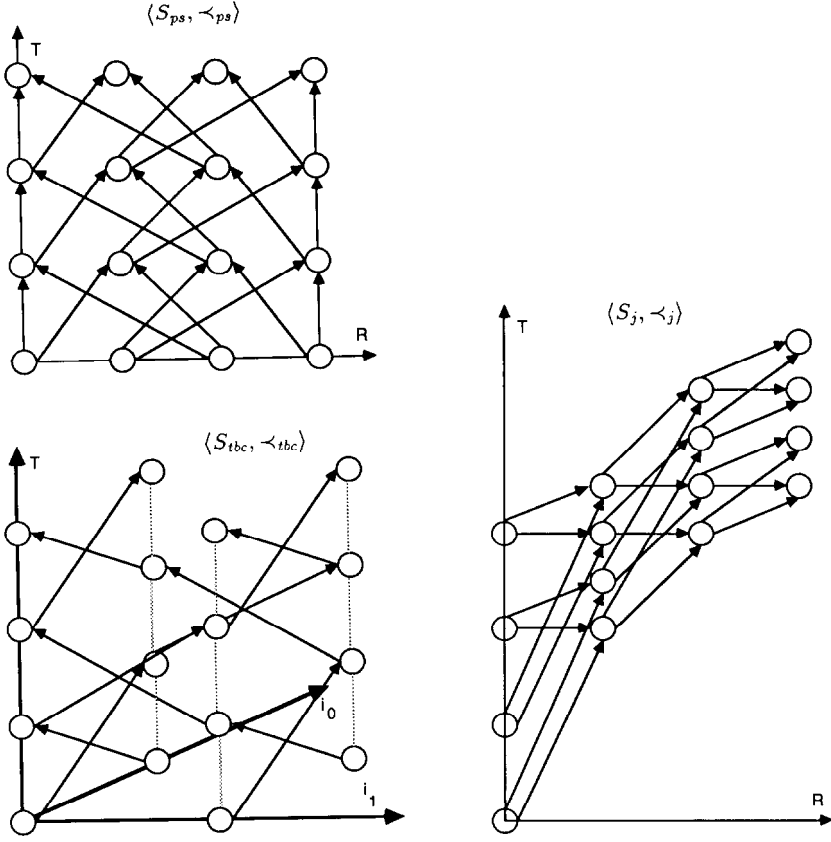
(Such a function exists. The condition above specifies a bijective restriction A'_j of A_j to a finite subset S' of S_{tbc} . The cardinality of $S_{\text{tbc}} \setminus S'$, $|\mathbb{N}|$, is the same as the cardinality of $S_j \setminus A'_j(S')$, so A'_j can be extended to a bijection $S_{\text{tbc}} \rightarrow S_j$.) Now $<_j$ is defined by $\text{fi}(\langle S_{\text{tbc}}, <_{\text{tbc}} \rangle, \langle S_j, <_j \rangle, A_j)$. By lemma 6.5, $<_j$ exists and it is easily shown to be a communication ordering. By definition A_j is $<_{\text{tbc}}$ -allowed (cf. Fig. 5).

We now show $\langle S_{\text{ps}}, <_{\text{ps}} \rangle \leftrightarrow^{\text{stt}} \langle S_j, <_j \rangle$:

$\langle S_j, <_j \rangle \leftrightarrow^{\text{stt}} \langle S_{\text{tbc}}, <_{\text{tbc}} \rangle$: Directly from Proposition 7.12 and the definition of $<_j$.

$\langle S_{\text{tbc}}, <_{\text{tbc}} \rangle \leftrightarrow^{\text{stt}} \langle S_{\text{ps}}, <_{\text{ps}} \rangle$: Let A_{ps} be a bijection $S_{\text{tbc}} \rightarrow S_{\text{ps}}$ such that the following holds: $i_0, \dots, i_{m-1} \in \{0, 1\}, t = 0, \dots, m+1$:

$$A_{\text{ps}}(t, i_{m-1}, \dots, i_0) = \left\langle t, \sum_{n=0}^{m-1} i_{n-t+1 \bmod m} 2^n \right\rangle.$$


 Fig. 5. $\langle S_{tbc}, \prec_{tbc} \rangle, \langle S_{ps}, \prec_{ps} \rangle, \langle S_j, \prec_j \rangle, m = 2$.

It is easy to verify that the restriction of A_{ps} specified above is 1-1 and it can be shown that this restricted function can be extended to a bijection $S_{tbc} \rightarrow S_{ps}$ in the same way as for A_j . Next, define $\prec_{A_{ps}}$ on S_{ps} by $\text{fi}(\langle S_{tbc}, \prec_{tbc} \rangle, \langle S_{ps}, \prec_{A_{ps}} \rangle, A_{ps})$. Let us show that $\prec_{A_{ps}} = \prec_{ps}$: From the definitions of \prec_{tbc} , A_{ps} it follows that

$$\left\langle t-1, \sum_{n=0}^{m-2} i_{n-t+2 \bmod m} 2^n + j_{m-t+1 \bmod m} 2^{m-1} \right\rangle \prec_{A_{ps}} \left\langle t, \sum_{n=0}^{m-1} i_{n-t+1 \bmod m} 2^n \right\rangle$$

for all relevant values of t . Let $i = \sum_{n=0}^{m-1} i_{n-t+1 \bmod m} 2^n$. Then

$$\sum_{n=0}^{m-2} i_{n-t+2 \bmod m} 2^n = \lfloor \frac{1}{2} i \rfloor \quad \text{and} \quad \langle t-1, \lfloor \frac{1}{2} i \rfloor + j_{m-t+1 \bmod m} 2^{m-1} \rangle \prec_{A_{ps}} \langle t, i \rangle.$$

It is now easy to check that really $\prec_{A_{ps}} = \prec_{ps}$. Thus, by Proposition 7.12, $\langle S_{tbc}, \prec_{tbc} \rangle \leftrightarrow^{\text{stt}} \langle S_{ps}, \prec_{ps} \rangle$, and transitivity gives $\langle S_j, \prec_j \rangle \leftrightarrow^{\text{stt}} \langle S_{ps}, \prec_{ps} \rangle$.

So all sets of cell actions (multiple-assignments) executable on any of the three structures are executable on the other two as well, and the mappings A_j , A_{ps} (and inverses) can be used to transform the schedules. Let us now schedule an FFT

algorithm, computing the discrete Fourier transform $\sum_{k=0}^{N-1} a(k) \omega^{jk}$, $j = 0, \dots, N-1$, $\omega = e^{2\pi i/N}$, $N = 2^m$, on the time-dependent binary $(m-1)$ -cube:

Recurrence relation (cf. [25], j, k are expanded into binary representation):

$i_0, \dots, i_{m-1} \in \{0, 1\}$:

$$b_0(i_{m-1}, \dots, i_0) \leftarrow a\left(\sum_{n=0}^{m-1} i_n 2^n\right);$$

$s \in \{1, \dots, m\}$:

$$b_s(i_{m-1}, \dots, i_{m-s}, \dots, i_0) \leftarrow \sum_{k_{m-s} \in \{0, 1\}} b_{s-1}(i_{m-1}, \dots, i_{m-s+1}, k_{m-s}, i_{m-s-1}, \dots, j_0) \omega^{(\sum_{n=0}^{s-1} i_{m-n-1} 2^n) k_{m-s} 2^{m-s}}.$$

Result: $b_m(i_{m-1}, \dots, i_0) = \sum_{k=0}^{N-1} a(k) \omega^{jk}$, $j = \sum_{n=0}^N i_{N-n} 2^n$ (binary reversed order)

This is an assignment representation of a set of cell actions for FFT. We map it into $\langle S_{\text{tbc}}, <_{\text{tbc}} \rangle$ (of order m) by a weakly correct mapping W as follows:

- the cell actions producing $b_0(i_{m-1}, \dots, i_0)$, $i_0 = 0, 1$ are mapped to $(0, i_{m-2}, \dots, i_1, i_{m-1})$;
- those producing $b_s(i_{m-1}, \dots, i_{m-s}, \dots, i_0)$, $s > 0$, $i_{m-s} = 0, 1$ are mapped to $(s, i_{m-1}, \dots, i_{m-s+1}, i_{m-s-1}, \dots, i_0)$.

Two cell actions are mapped onto every point in S_{tbc} . Thus we are running m -FFT on an $(m-1)$ -cube. The reason is efficiency; both cell actions use the same indata and have subexpressions in common which can be utilized to reduce hardware size.

Let us finally examine $<_{cW}$ on S_{tbc} to see that $<_{cW} \subseteq <_{\text{tbc}}$. $<_{cW}$ can be obtained by identifying input and output variables from cell actions, and the result is (renaming binary indices):

$i_0, \dots, i_{m-2}, k_{m-1} \in \{0, 1\}$:

$$(0, i_{m-2}, \dots, i_1, k_{m-1}) <_{cW} (1, i_{m-2}, \dots, i_0);$$

$s \in \{2, \dots, m\}$, $i_0, \dots, i_{m-2}, j_{m-s} \in \{0, 1\}$:

$$(s-1, i_{m-2}, \dots, j_{m-s}, \dots, i_0) <_{cW} (s, i_{m-2}, \dots, i_{m-s}, \dots, i_0).$$

It is easy to see that, in fact, $<_{cW} = <_{\text{tbc}}$. Therefore, this FFT algorithm is executable on all three space-time structures. $\langle S_j, <_j \rangle$ is especially interesting since it has an advantage with respect to wiring; all connections are local in space. An implementation of a slightly different FFT algorithm on a similar network, described in terms of delay elements and functional units, is given in [7], but the example here shows that *any* recursive partitioning-type algorithm that can be recursively described using binary indices and mapped to $\langle S_{\text{tbc}}, <_{\text{tbc}} \rangle$ as above can be scheduled on $\langle S_j, <_j \rangle$. This must, for instance, hold for bitonic sort.

Acknowledgment

I want to thank Stefan Arnborg for his helpful comments. This work was partially supported by the Swedish Board for Technical Development (STU).

References

- [1] G. Birkhoff and J.D. Lipson, Heterogenous algebras, *J. Combin. Theory* **8** (1970) 115–133.
- [2] P.K. Cappello and K. Steiglitz, Unifying VLSI array design with linear transformations of space-time, Research Rept. TRCS83-03, Dept. Computer Science, Univ. of California at Santa Barbara, 1983.
- [3] M.C. Chen, A synthesis method for systolic designs, Research Rept. YALEU/DCS/RR-334, Dept. Computer Science, Yale University, 1985.
- [4] M.C. Chen, Synthesizing systolic designs, Research Rept. YALEU/DCS/RR-374, Dept. Computer Science, Yale University, 1985.
- [5] K. Culik II and I. Fris, Topological transformations as a tool in the design of systolic networks, Rept. CS-84-11, Dept. Computer Science, University of Waterloo, 1984.
- [6] G. Grätzer, *Universal Algebra* (Springer, Berlin, 1979).
- [7] L. Johnsson and D. Cohen, A formal derivation of array implementation of FFT algorithms, in: *Proc. USC Workshop on VLSI and Modern Signal Processing* (1982) 53–63.
- [8] L. Johnsson, U. Weiser, D. Cohen and A.L. Davis, Towards a formal treatment of VLSI arrays, in: *Proc. Second Caltech Conf. on VLSI* (1981) 378–398.
- [9] D.J. Kuck, Y. Muraoka and S.C. Chen, On the number of operations simultaneously executable in Fortran-like programs and their resulting speedup, *IEEE Trans. Comput.* **C-21** (1972) 1293–1310.
- [10] H.T. Kung and C.E. Leiserson, Algorithms for VLSI processor arrays, in: C. Mead and L. Conway, *Introduction to VLSI Systems* (Addison-Wesley, Reading, MA, 1980) Chapter 8.3.
- [11] H.T. Kung and W.T. Lin, An algebra for VLSI algorithm design, Tech. Rept., Dept. Computer Science, Carnegie-Mellon Univ., PA, 1983.
- [12] L. Lamport, The parallel execution of DO loops, *Comm. ACM* **17** (1974) 83–93.
- [13] H. Lev-Ari, Modular computing networks: a new methodology for analysis and design of parallel algorithms/architectures, ISI Rept. 29, Integrated Systems Inc., 1983.
- [14] H. Lev-Ari, Canonical realizations of completely regular modular computing networks, ISI Rept. 41, Integrated Systems Inc., 1984.
- [15] B. Lisper, Description and synthesis of systolic arrays, Tech. Rept. TRITA-NA-8318, NADA, KTH, Stockholm, 1983.
- [16] B. Lisper, Hardware synthesis from specification with polynomials, Tech. Rept. TRITA-NA-8506, NADA, KTH, Stockholm, 1985.
- [17] B. Lisper, Synthesizing synchronous systems by static scheduling in space-time, Ph.D. Thesis, TRITA-NA-8701, NADA, KTH, Stockholm, 1987.
- [18] Z. Manna and R. Waldinger, *The Logical Basis for Computer Programming, Volume II: Deductive Systems* (Addison-Wesley, Reading, MA, to appear).
- [19] Z. Manna and R. Waldinger, *The Logical Basis for Computer Programming, Volume I: Deductive Reasoning* (Addison-Wesley, Reading, MA, 1985).
- [20] W.L. Miranker and A. Winkler, Spacetime representations of computational structures, *Computing* **32** (1984) 93–114.
- [21] D.T. Moldovan, On the analysis and synthesis of VLSI algorithms, *IEEE Trans. Comput.* **C-31** (1982) 1121–1126.
- [22] Y. Muraoka, Parallelism exposure and exploitation in programs, Ph.D. Thesis, Dept. Computer Science, Univ. of Illinois at Urbana-Champaign, 1971.
- [23] M. Nielsen, G. Plotkin and G. Winskel, Petri nets, event structures and domains, Part I, *Theoret. Comput. Sci.* **13** (1981) 85–108.
- [24] I.V. Ramakrishnan, D.S. Fussell and A. Silberschatz, Towards a characterization of programs for a model of VLSI array-processors, Tech. Rept. TR-202, Dept. Computer Science, Univ. of Texas at Austin, 1982.
- [25] H.S. Stone, Parallel processing with the perfect shuffle, *IEEE Trans. Comput.* **C-20** (1971) 153–161.
- [26] U. Weiser and A.L. Davis, A wavefront tool for VLSI design, in: H.T. Kung, B. Sproull and G. Steele, eds., *VLSI Systems and Computations* (Springer, Berlin, 1981) 226–234.